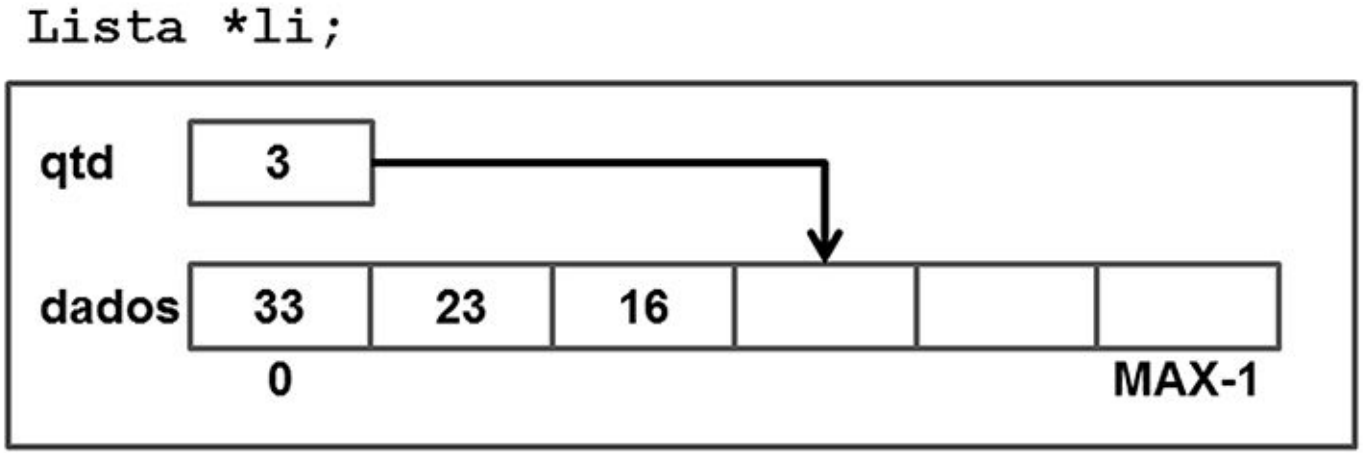
Pilhas

Sergio Canuto
sergio.canuto@ifg.edu.br

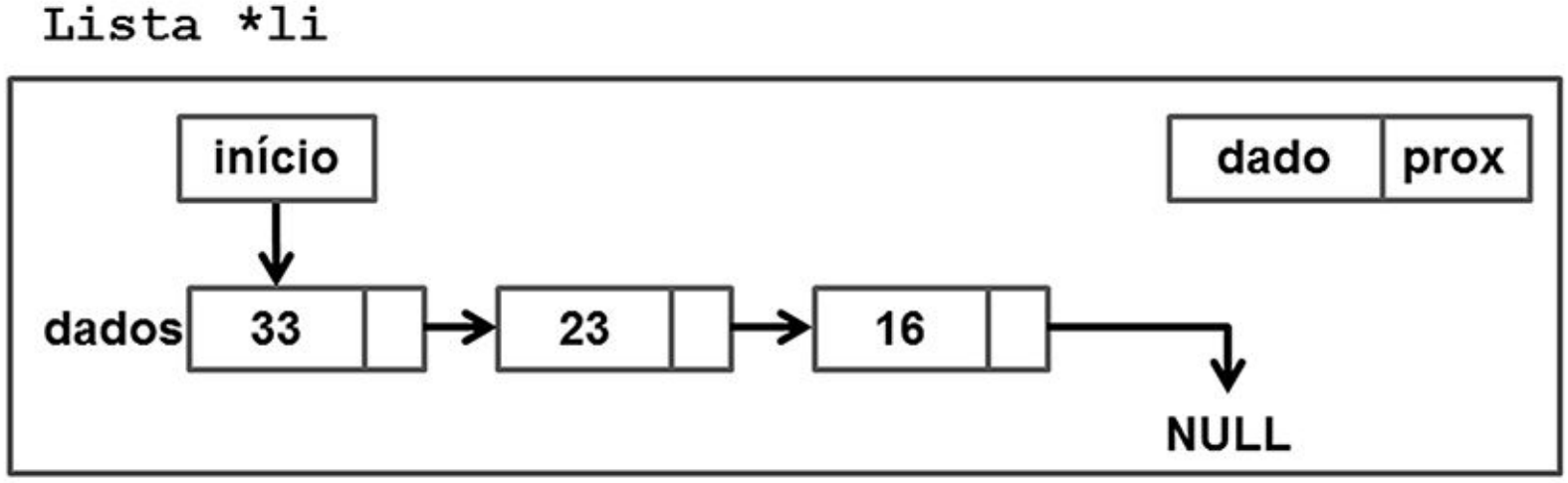
Relembrando...

- **LISTA SEQUENCIAL ESTÁTICA**
  - Uma **lista sequencial estática** ou **lista linear estática** é uma lista definida utilizando alocação estática e acesso sequencial dos elementos



LISTA DINÂMICA ENCADEADA

- Uma **lista dinâmica encadeada** é uma lista definida utilizando alocação dinâmica e acesso encadeado dos elementos. Cada elemento contém:
  - um campo de **dado**, utilizado para armazenar a informação.
  - um campo **prox**, ponteiro que indica o próximo elemento na lista.

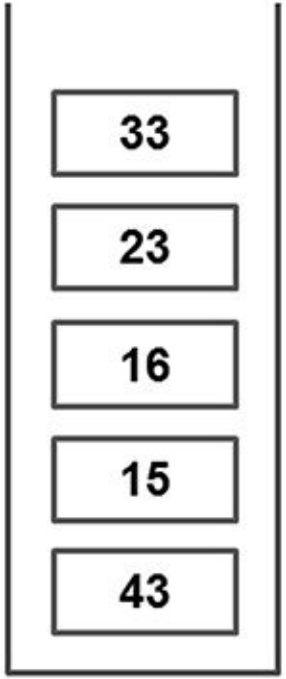


Pilhas - Definição

- O conceito de pilha é algo bastante comum:
  - Somente podemos inserir um novo item na pilha se o colocarmos acima dos demais e apenas removeremos o item que está no topo da pilha.
  - As pilhas são implementadas e se comportam de modo muito similar às listas, sendo, muitas vezes, consideradas um tipo especial de lista em que a inserção e a remoção são realizadas sempre na mesma extremidade.
  - Conhecidas como estruturas do tipo último a entrar, primeiro a sair ou LIFO (Last In First Out): os elementos são removidos da pilha na ordem inversa daquela em que foram inseridos.

Pilhas - Definição

Pilha



Pilhas - Implementações

- Existem dois tipos de implementações principais para uma Pilha:
  - **Alocação estática com acesso sequencial**: o espaço de memória é alocado no momento da compilação do programa, ou seja, é necessário definir o número máximo de elementos que a pilha irá possuir. Desse modo, os elementos são armazenados de forma consecutiva na memória (como em um array ou vetor) e a posição de um elemento pode ser facilmente obtida a partir do início da pilha.
  - **Alocação dinâmica com acesso encadeado**: o espaço de memória é alocado em tempo de execução, ou seja, a pilha cresce à medida que novos elementos são armazenados, e diminui à medida que elementos são removidos. Nessa implementação, cada elemento pode estar em uma área distinta da memória, não necessariamente consecutivas. É necessário então que cada elemento da pilha armazene, além da sua informação, o endereço de memória onde se encontra o próximo elemento. Para acessar um elemento, é preciso percorrer todos os seus antecessores na pilha.

Pilhas - Operações básicas

Independentemente do tipo de alocação e acesso usado na implementação de uma pilha, as seguintes operações básicas são sempre possíveis:

- Criação da pilha.
- Inserção de um elemento no topo da pilha.
- Remoção de um elemento do topo da pilha.
- Acesso ao elemento do topo da pilha.
- Destruição da pilha.
- Além de informações com tamanho, se a pilha está cheia ou vazia.

PILHA SEQUENCIAL ESTÁTICA

- Uma **pilha sequencial estática** ou **pilha linear estática** é uma pilha definida utilizando alocação estática (vetor)
  - Tipo mais simples de pilha possível.
  - Definida utilizando um array, de modo que o sucessor de um elemento ocupa a posição física seguinte deste.
  - Além do array, essa pilha utiliza um campo adicional (**qtd**) que serve para indicar o quanto do array já está ocupado pelos elementos (**dados**) inseridos na pilha.

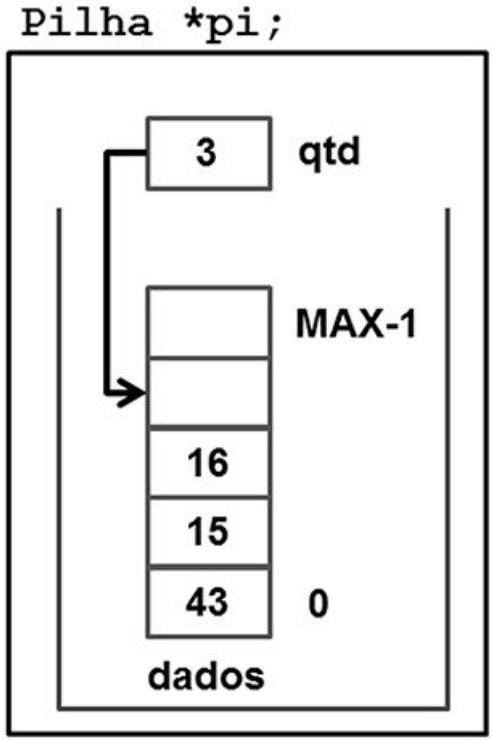
Vantagem:

- A principal vantagem de se utilizar um array na definição de uma **pilha sequencial estática** é a facilidade de criar e destruir a pilha.

Desvantagem:

- A principal desvantagem é a necessidade de definir previamente o tamanho do array e, consequentemente, da pilha.

PILHA SEQUENCIAL ESTÁTICA



TAD da Pilha

Arquivo PilhaSequencial.h

```
01  #define MAX 100
02  struct aluno{
03      int matricula;
04      char nome[30];
05      float n1,n2,n3;
06  };
07  typedef struct pilha Pilha;
08
09  Pilha* cria_Pilha();
10  void libera_Pilha(Pilha* pi);
11  int acessa_topo_Pilha(Pilha* pi, struct aluno *al);
12  int insere_Pilha(Pilha* pi, struct aluno al);
13  int remove_Pilha(Pilha* pi);
14  int tamanho_Pilha(Pilha* pi);
15  int Pilha_vazia(Pilha* pi);
16  int Pilha_cheia(Pilha* pi);
```

Arquivo PilhaSequencial.c

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  #include "PilhaSequencial.h" //inclui os protótipos
04  //Definição do tipo Pilha
05  struct pilha{
06      int qtd;
07      struct aluno dados[MAX];
08  };
```

TAD da Pilha

Criando uma pilha

```
01  Pilha* cria_Pilha(){
02      Pilha *pi;
03      pi = (Pilha*) malloc(sizeof(struct pilha));
04      if(pi != NULL)
05          pi->qtd = 0;
06      return pi;
07  }
```

Destruindo uma pilha

```
01  void libera_Pilha(Pilha* pi){
02      free(pi);
03  }
```

Tamanho da pilha

```
01  int tamanho_Pilha(Pilha* pi){
02      if(pi == NULL)
03          return -1;
04      else
05          return pi->qtd;
06  }
```

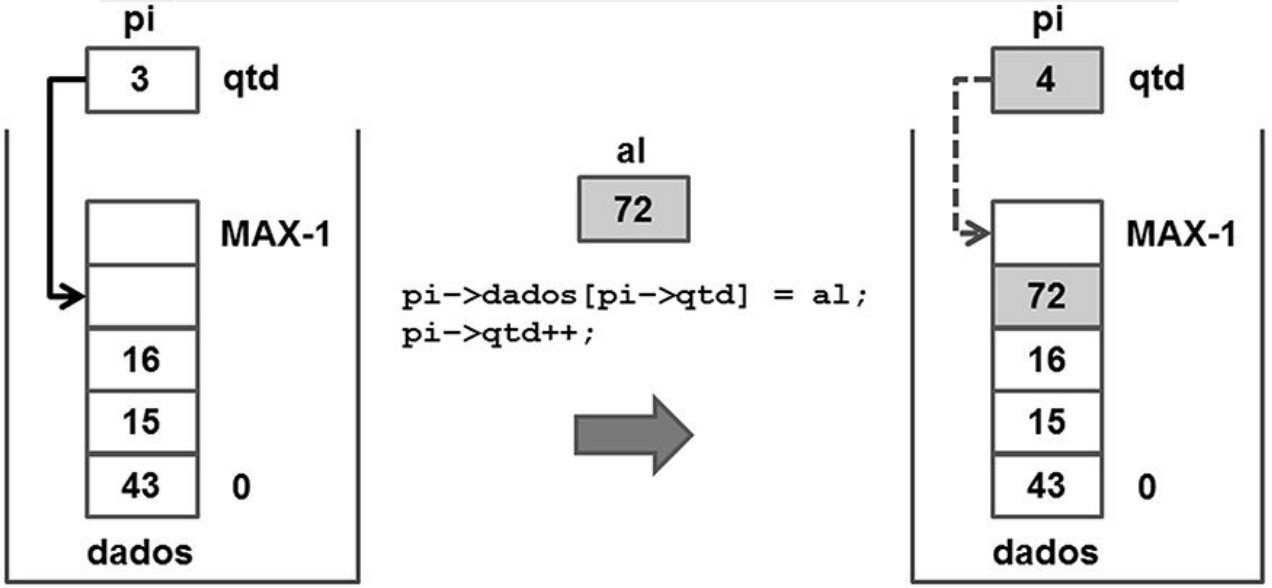
Retornando se a pilha está vazia

```
01  int Pilha_vazia(Pilha* pi){
02      if(pi == NULL)
03          return -1;
04      return (pi->qtd == 0);
05  }
```

TAD da Pilha - Inserção

Inserindo um elemento na pilha

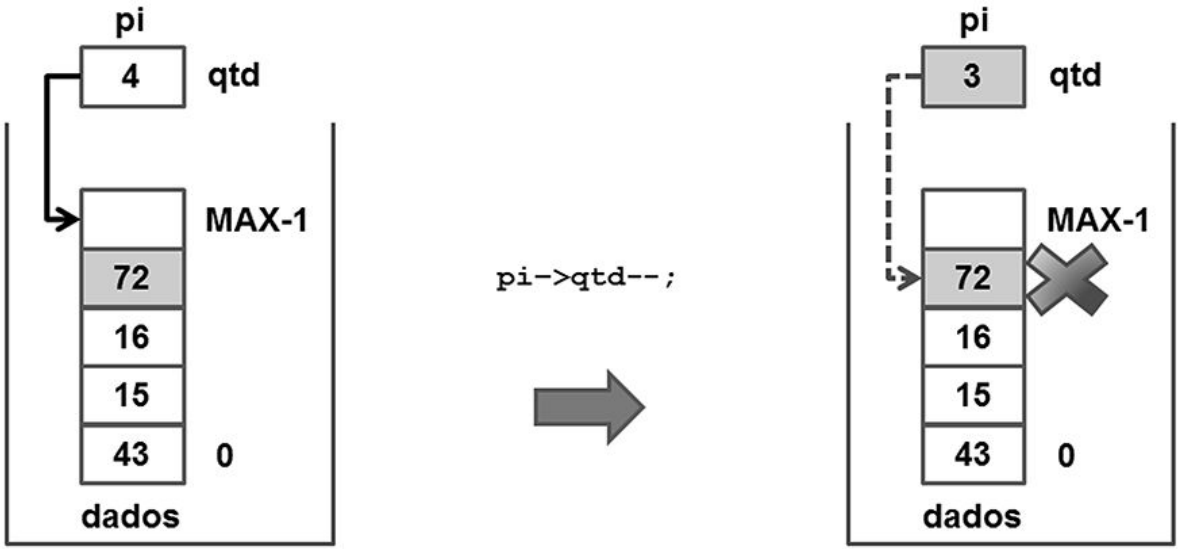
```
01  int insere_Pilha(Pilha* pi, struct aluno al){
02      if(pi == NULL)
03          return 0;
04      if(pi->qtd == MAX)//pilha cheia
05          return 0;
06      pi->dados[pi->qtd] = al;
07      pi->qtd++;
08      return 1;
09  }
```



TAD da Pilha - Remoção

Removendo um elemento da pilha

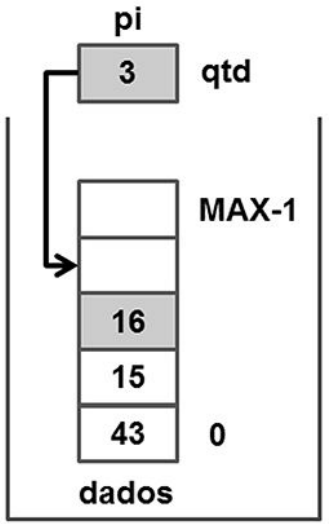
```
01  int remove_Pilha(Pilha* pi){
02      if(pi == NULL || pi->qtd == 0)
03          return 0;
04      pi->qtd--;
05      return 1;
06  }
```



TAD da Pilha - Acesso

Acessando o topo da pilha

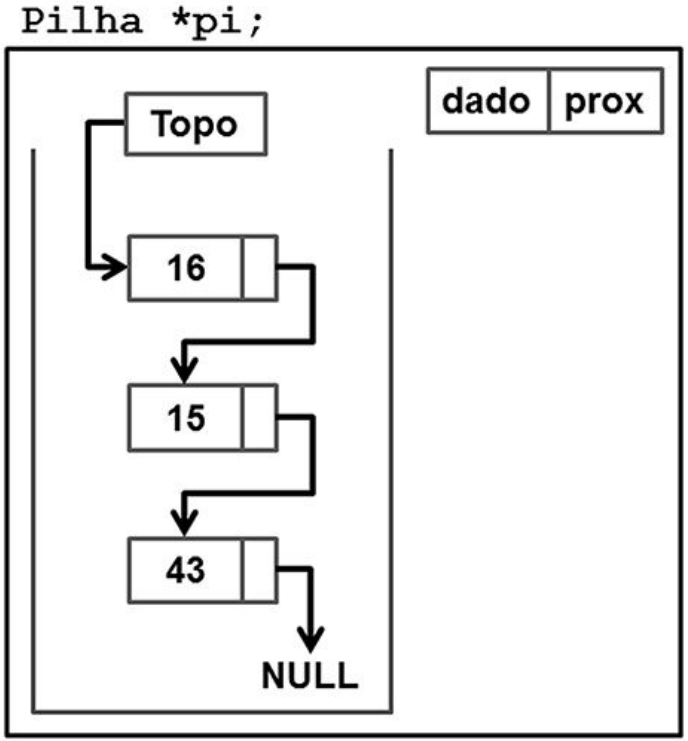
```
01  int acessa_topo_Pilha(Pilha* pi, struct aluno *al){
02      if(pi == NULL || pi->qtd == 0)
03          return 0;
04      *al = pi->dados[pi->qtd-1];
05      return 1;
06  }
```



Acesso :

```
*al = pi->dados[pi->qtd -1];
```

Pilhas - Pilha Dinâmica encadeada



Pilha Dinâmica encadeada

Arquivo PilhaDin.h

```
01  struct aluno{
02      int matricula;
03      char nome[30];
04      float n1,n2,n3;
05  };
06  typedef struct elemento* Pilha;
07
08  Pilha* cria_Pilha();
09  void libera_Pilha(Pilha* pi);
10  int acessa_topo_Pilha(Pilha* pi, struct aluno *al);
11  int insere_Pilha(Pilha* pi, struct aluno al);
12  int remove_Pilha(Pilha* pi);
13  int tamanho_Pilha(Pilha* pi);
14  int Pilha_vazia(Pilha* pi);
15  int Pilha_cheia(Pilha* pi);
```

Arquivo PilhaDin.c

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  #include "PilhaDin.h" //inclui os protótipos
04  //Definição do tipo Pilha
05  struct elemento{
06      struct aluno dados;
07      struct elemento *prox;
08  };
09  typedef struct elemento Elem;
```

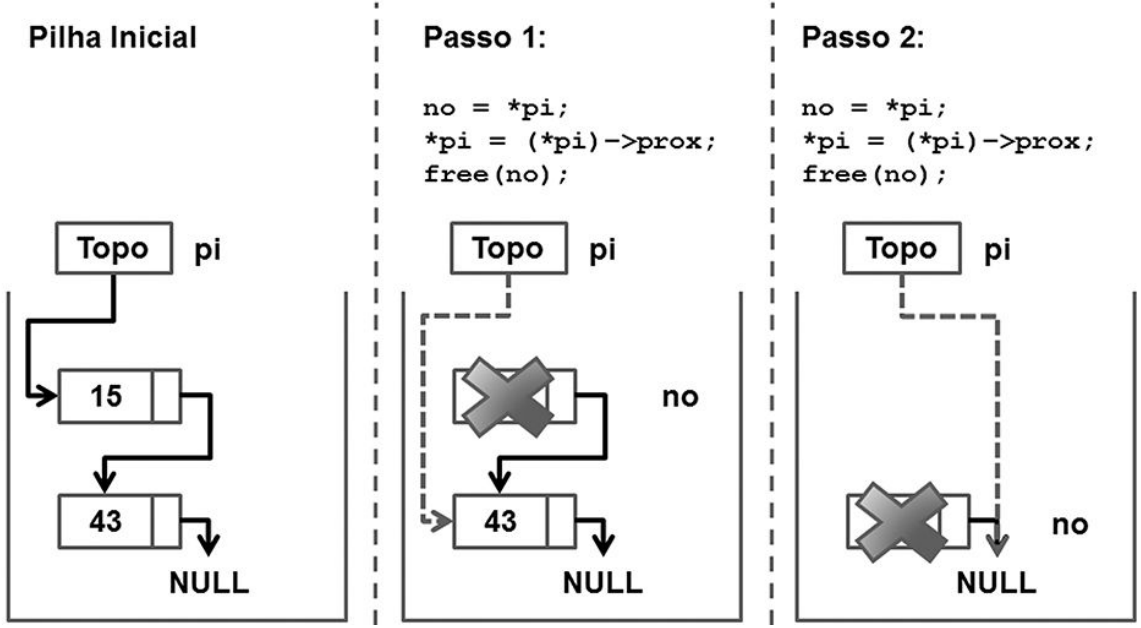
Pilha Dinâmica encadeada

Criando uma pilha

```
01  Pilha* cria_Pilha(){
02      Pilha* pi = (Pilha*) malloc(sizeof(Pilha));
03      if(pi != NULL)
04          *pi = NULL;
05      return pi;
06  }
```

Destruindo uma pilha

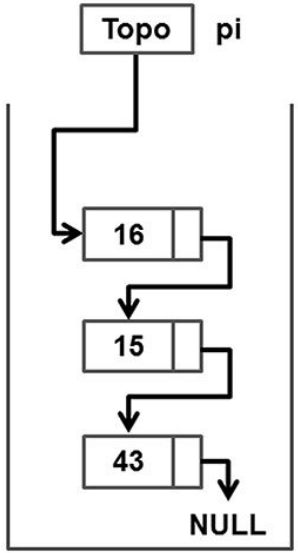
```
01  void libera_Pilha(Pilha* pi){
02      if(pi != NULL){
03          Elem* no;
04          while((*pi) != NULL){
05              no = *pi;
06              *pi = (*pi)->prox;
07              free(no);
08          }
09          free(pi);
10      }
11  }
```

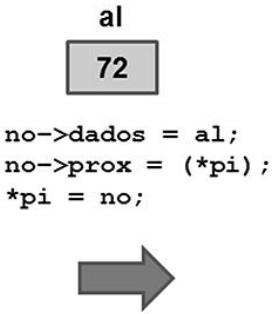


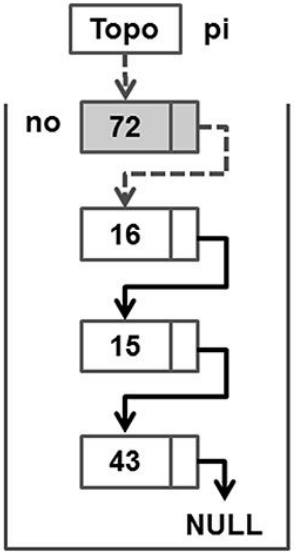
Pilha Dinâmica encadeada

Inserindo um elemento na pilha

```
01  int insere_Pilha(Pilha* pi, struct aluno al){
02      if(pi == NULL)
03          return 0;
04      Elem* no;
05      no = (Elem*) malloc(sizeof(Elem));
06      if(no == NULL)
07          return 0;
08      no->dados = al;
09      no->prox = (*pi);
10      *pi = no;
11      return 1;
12  }
```



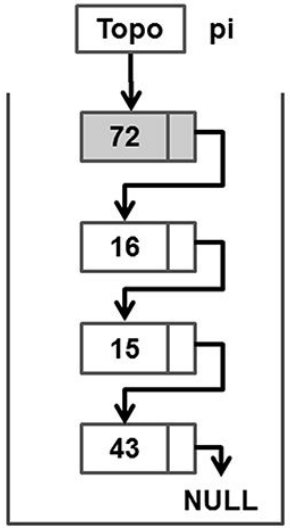




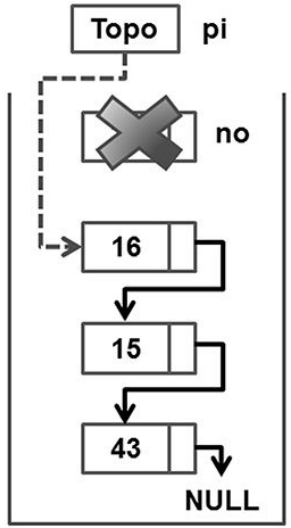
Pilha Dinâmica encadeada

Removendo um elemento da pilha

```
01  int remove_Pilha(Pilha* pi){
02      if(pi == NULL)
03          return 0;
04      if((*pi) == NULL)
05          return 0;
06      Elem *no = *pi;
07      *pi = no->prox;
08      free(no);
09      return 1;
10  }
```




```
Elem *no = *pi;
*pi = no->prox;
free(no);
```



Pilhas - Operações básicas

- As pilhas são consideradas um tipo especial de lista em que a inserção e remoção são realizadas sempre na mesma extremidade da lista.
- Podemos concluir, então, que uma pilha nada mais é do que uma lista sujeita a uma ordem de entrada e saída.
- Sendo assim, podemos implementar uma pilha utilizando uma lista

Atividade: pilhas e filas

1) O desenvolvimento de um software que analisa bases de DNA, representadas pelas letras A, C, G, T, utilizou-se as estruturas de dados: pilha e fila. Considere que, se uma sequência representa uma pilha, o topo é o elemento mais à esquerda; e se uma sequência representa uma fila, a sua frente é o elemento mais à esquerda. Analise o seguinte cenário: "a sequência inicial ficou armazenada na primeira estrutura de dados na seguinte ordem: (A, G, T, C, A, G, T, T). Cada elemento foi retirado da primeira estrutura de dado se inserido na segunda estrutura de dados, e a sequência ficou armazenada na seguinte ordem: (T, T, G, A, C, T, G, A). Finalmente, cada elemento foi retirado da segunda estrutura de dados e inserido na terceira estrutura de dado e a sequência ficou armazenada na seguinte ordem: (T, T, G, A, C, T, G, A)". Qual a única sequência de estruturas de dados apresentadas a seguir pode ter sido usada no cenário descrito acima? Justifique a resposta.

a) Pilha - Pilha - Pilha

b) Fila - Fila - Pilha.

c) Fila - Pilha - Pilha.

d) Pilha - Fila - Pilha.

e) Fila - Pilha - Fila.

Atividade: pilhas e filas

1) Baixe o arquivo https://scanuto.com/ex_pilha.zip e complete a função “inverte_pilha”, invertendo o conteúdo da pilha de entrada em uma nova pilha. Utilize as funções cria_Pilha, consulta_topo_Pilha, insere_Pilha, remove_Pilha e libera_Pilha.

Atividade:

2) Explique o código que implementa uma pilha sequencial estática (sem TAD). Há algum erro no código?

```
1   #define N 100
2   #include <stdio.h>
3   #include <stdlib.h>
4
5   static char pilha[N];
6   static int t;
7
8   void iniciapilha (void) {
9      t = 0;
10  }
11
12  void empilha (char y) {
13     pilha[t] = y;
14     t++;
15  }
16  char desempilha (void) {
17     t=t-1;
18     return pilha[t+1];
19  }
20
21  int pilhavazia (void) {
22     return t <= 0;
23  }
24
25  int pilhacheia (void) {
26     return t > N;
27  }
```

Atividade:

3) Implemente uma função que receba uma string, empilhe cada caractere na pilha, e imprima a string invertida, caractere por caractere, usando a função desempilha (utilize a correção do exercício anterior). *Em C, o último caractere da string declarada é o '\0'.

```
1   #define N 100
2   #include <stdio.h>
3   #include <stdlib.h>
4
5   static char pilha[N];
6   static int t;
7
8   void iniciapilha (void) {
9      t = 0;
10  }
11
12  void empilha (char y) {
13     pilha[t] = y;
14     t++;
15  }
16  char desempilha (void) {
17     t=t-1;
18     return pilha[t+1];
19  }
20
21  int pilhavazia (void) {
22     return t <= 0;
23  }
24
25  int pilhacheia (void) {
26     return t > N;
27  }
28
29  void imprime_invertido(char* str){
30  //seu codigo aqui!
31  }
32  int main(){
33  char str_entrada[]=”inverta”;
34
35  imprime_invertido(str_entrada);
36  return 0;
37  }
```

Atividade:

4) Implemente a função checa_parenteses abaixo que recebe uma string contendo uma expressão aritmética e retorna se ela está com a parentização correta. A função deverá verificar se cada “abre parênteses’” tem um “fecha parênteses” correspondente. Utilize as funções de empilha, desempilha e pilhavazia para fazer a verificação eficiente com a função. Se correto, retorna 1:

int checa_parenteses(char* string_parenteses)

•Correto:

(())

((()))

()()

•Incorreto:

)(

((()(

))((

Atividade:

5) Implemente a função checa_palindromo abaixo que recebe uma palavra e retorna se ela é ou não um palíndromo. Utilize as funções de empilha, desempilha e pilhavazia para fazer a verificação eficiente com a função. Se correto, retorna 1:

int checa_palindromo(char* string_palindromo)

Ex. de palíndromos:

- osso
- arara
- ele

Referências

Estrutura de Dados descomplicada em Linguagem C (André Backes): Cap 8;

Vídeo aulas (31-44):
https://programacaodescomplicada.wordpress.com/indice/estrutura-de-dados/

Implementações:

http://www.facom.ufu.br/~backes/wordpress/